Algorithm Selection using Behavioral Synthesis

Miodrag Potkonjak
Dept. of Computer Science, University of California, Los Angeles

Jan Rabaey
Dept. of EECS, University of California, Berkeley

HYPER is a third generation high level synthesis system, targeted at numerically intensive applications. By shifting the emphasis from the traditional high level synthesis tasks (such as scheduling, assignment, allocation and module selection) to the domain of optimizing transformations, new avenues for high level synthesis are opened. One of the most exciting among them, which probably has the largest impact on the quality of the design, is to select and optimize the algorithms for a given application.

The paper starts with a brief overview of the HYPER system, with special stress on the optimizing transformation methodology. After this, the paper concentrates on the exploration of the algorithmic design space. We show how HYPER can be used to guide and conduct a proper algorithmic selection process and how transformations in HYPER can improve the performance or cost of real life applications with orders of magnitude.

INTRODUCTION

Motivation: Why yet another higher level synthesis system?

High level synthesis has emerged as one of the most exciting and most challenging areas in CAD. It is both eagerly and reluctantly expected by the design community. Eagerly since it promises an increase in productivity, it opens new venues for designers and it relieves them of many mundane and tedious tasks, which are currently undertaken manually; reluctantly since it dictates a dramatic and drastic change in the way the design process is conducted. We have currently reached the point, where high level synthesis has to prove itself as one the most important and most influential CAD tools in design, or will slowly disappear as an exciting, but also dead end academic research avenue.

While, of course, its destiny will ultimately be determined by the effectiveness and efficiency of the tools in industrial designs and their viability to compete with the manual design process, it is very instructive to look at the genealogy of high level synthesis. This approach enables us to fully leverage on already achieved successes and achievements and avoid some of the dangerous pitfalls, which prevented it from becoming a standard engineering tool at present.

The identification of different generations in technological developments is an uncertain undertaking and a classification according to different criteria appears often to be more natural. However, it seems that until recently there have been two major generations of high level synthesis systems. The first generation was developed by the researchers from the computer architecture community [Bar73]. Although initially targeted at RTL level synthesis, the focus switched later to high level synthesis. Many important results were obtained and almost all high level synthesis tasks were precisely defined and outlined; these include allocation, assignment, scheduling, module selection, partitioning, transformations and interface design. Early researchers in the field accurately identified many most relevant issues and proposed the first approaches to address them. For example, the first papers already stressed the importance of design space exploration, which just recently has come again into the attention of both the high level synthesis and design communities. However, the supporting algorithmic and computing environments were of insufficient capability to support overly ambitious design tasks, such as microprocessor design. In the last phase of the first generation, great hopes were based on the use of artificial intelligence techniques. The success was limited, however. At that point, a majority of researchers in high level synthesis field reached a consensus, that, without dramatic improvements in knowledge acquisition and manipulation techniques, automatic competitive design of general purpose computers is unlikely. This conclusion lead to the second generation of high level synthesis, which targets more restricted design areas and synthesis problems.

The second generation is characterized by a more focused approach and greater attention to the solving of a few high level synthesis tasks. More than a hundred high level synthesis systems have been reported in the CAD, DSP, communication and circuits and systems literature [Wal91]. Despite the large number of systems and the variety of the application domains and the employed optimization techniques, there are striking similarities among many of them: the focus of the synthesis process is almost exclusively on the scheduling, allocation and assignment tasks, although more recently partitioning and module selection have been drawing attention as well.

While providing a clear insight in the nature of the problems and producing a wide variety of excellent solutions, the idea of focusing exclusively on isolated problems generates some monumental pitfalls. Tools were tested on a small set of academic benchmarks and optimized to outperform previously published approaches ("the fifth order elliptical filter syndrome"). No attempts are made to validate the approaches and to assess their potential on real life examples. As a result, a clear vision of the actual cost of a design was lost. The cost of a design is rarely determined by the number of execution units (the "primary" optimization goal), but most often is set by the "secondary" resources, such as registers, interconnect, control and input/output. Although background memory quite often dominates the cost of a system [DeM92, Pat90], it was only recently addressed in a few papers. Finally, while hierarchy has long been accepted as one of the keys to successful system design, it has only drawn marginal attention in the synthesis community.

Compared with the effort spent on individual tasks, scant attention was paid to tool integration. We refer here not so much to software integration, but mainly to the way tools interact and communicate with each other and with the environment. Information feedback to the designer is limited, if existing. The relationship of high level synthesis with respect to the overall system design problem is ill understood or not addressed at all. Designers have, however, made clear that synthesis only makes sense when seen in a system context and in co-existence with other compilation tools, such as available or under development for programmable hardware and heterogeneous hardware/software platforms [Pau92]. Finally, the organization of current synthesis tools is such that they offer scarce opportunity to address the requirements of the next generation of CAD tools, such as computer aided algorithm selection and design

In depth reviews of the state of the art in high level synthesis can be found in [McF90, Wol91b]. For an inspirational and fresh look at the current trends in high level synthesis and in CAD in general, please refer to [DeM92].

Notwithstanding the above observations, several systems have been designed, which transcend those problems and are successful in designing complex pilot designs. A number of them are currently making it to the commercial arena. The response from the design community is still muted, however. Even if a system succeeds to proceed smoothly from a behavioral level description of an application to a layout, designers complain that the system does not offer any added value: at most, it performs a number of tasks they knew how to do anyhow and it does not offer any help in the most demanding design problems, such as the exploration of the architectural space. So, while the feasibility proof of concept has been demonstrated, the establishment of high level synthesis as a dominant or attractive design option is still pending. If high level synthesis wants to avoid the destiny of other research fields, which generated great expectations and even greater disappointments, it is essential that a number of those observations are successfully addressed.

The HYPER high level synthesis system tries to address some of those issues and is based on two major thrusts:

1) The only way high level synthesis can get an edge over current design approaches, is by identifying and treating optimization intensive tasks, which are difficult (or even impossible) to address manually due to the computational and logical complexity. We believe that optimizing transformations fall into that category. They help to transcend the limitations of the initial specification and can result in more dramatic reductions of the implementation cost (be it area, speed or power) than any other synthesis task. Closely related to transformations is the estimation task, which helps to plot the design space and serves as an essential feedback to the designer. Finally, transformations are the key link to the next layer in the design synthesis process, being algorithm design and selection. The latter is the main topic of this paper. It is important to notice that the use of algorithm selection and optimization use has not been previously addressed in the high level synthesis literature.

2) The only other high level synthesis task, which can have even more dramatic influence on the overall quality of a design, is, surprisingly, the way the high level synthesis tools are organized and integrated. Although it is easy to notice that all high (and low) level synthesis tasks are interrelated, the majority of current day high level synthesis systems concentrate heavily on

individual tasks. Integrating these tools in an effective manner requires a global view on the goal to be achieved, which can only be offered by estimations. The HYPER synthesis manager has been described elsewhere in a detail [Rab90, Pot91a] and is out of the scope of this paper.

Paper Organization

The rest of the paper is organized in the following way. First, the HYPER system and its main components are briefly surveyed. Next, transformations and their use in high level synthesis and in the HYPER system in particular are discussed. Following section contains the most important material in the paper: how transformations can be used to efficiently and effectively select the best computational structure for a given behavioral description and set of goals and constraints. The detailed study of a real life example is presented in this section.

Finally, we conclude by outlining the issues which we consider as the most important directions in high level synthesis in general and in HYPER in particular.

HYPER

Real time applications in areas such as terrestrial, mobile and satellite communications, speech, image and video processing, radar, sonar and computerized tomography often require high performance dedicated datapaths. The structure of the datapath is strongly correlated to the structure of the computations. The controller, if existing at all, is small and contains at most a few tens of states. Pipelining is often used to improve throughput performance.

The synthesis of this type of architecture is an involved, meticulous and cumbersome process, which implies a strong need for sophisticated CAD tool support. The HYPER system addresses exactly this class of numerically intensive algorithms [Rab91]. A detailed description of the HYPER system and its toolset is given elsewhere [Rab91, Pot91a, Pot92b, Chu92, Hoa92]. We will limit our discussion to a presentation of the overall strategy and organization and a review of the transformational techniques.

The software organization of HYPER is shown in Figure 1. The input to the system is a description of application in a behavioral language, called Silage [Hil92]. Silage is an applicative signal-flow language which is particularly effective for the specification of digital signal processing algorithms. The Silage description is translated into an intermediate CDFG (control data

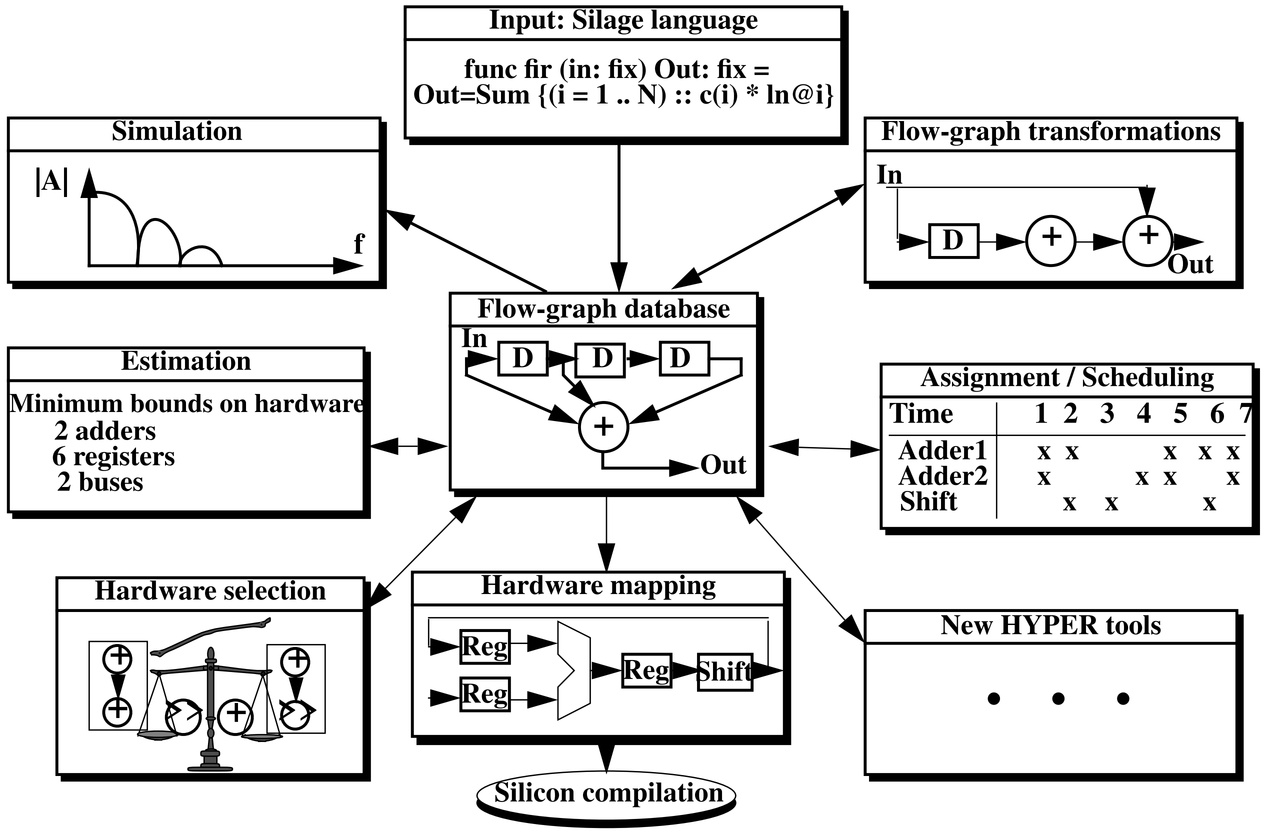


FIGURE 1: Overview of The HYPER System

flow graph) format, which serves as the central repository, on which all synthesis tools are operating and all results are annotated. The tool library includes tasks, such as simulation, hardware module selection, transformations, allocation, assignment, scheduling and background memory management. The single most important future of HYPER is the synthesis manager program, which supports the exploration of the design space using classical high level synthesis tools, transformations and a set of estimation and feedback information tools. The synthesis manager uses a single global design quality measure, called the resource utilization, to direct the design space exploration. It also guides an ergonomic user interface, which facilitates user interactivity.

Once a final result is obtained, HYPER maps the annotated CDFG into an architectural description, which is translated into layout using a silicon compiler. Currently, the Lager IV [Bro92] is used, but a VHDL interface to commercial RTL synthesis environments is provided as well.

Transformations in HYPER

Transformations: What? Why? How?

As we already mentioned, the single most important feature of HYPER is the synthesis manager program. Although, the synthesis manager is capable of achieving implementations, which are optimal or close to optimal (according to the min-bounds predicted by the estimations), its power to compete with and outperform a human designer heavily depends on transformations. Transformations are alternations of the structure of a computation such that the behavior (the relationship between output and input data) is preserved. The impetus behind the application of transformations is the hope that the transformed computational structure might result in an implementation with improved performance. As mentioned above, the performance measure has many dimensions such as area, throughput, power, testability, fault-tolerance and I/O requirements.

A more insightful definition is that transformations are elegant algebraic identities and correctness preserving reorganizations of the computational flow. This definition, as we will show later, is an excellent starting point to answer a number of important questions about transformations, and their role in high level synthesis. Those questions include the following:

(1) Which transformations to implement and support in a high level synthesis system?

It has been recognized for a long time in the compiler community, and now becomes obvious in high level synthesis community too, that implementing and supporting a large set of transformations is a rewarding, but also tedious and overly meticulous effort [Wol92]. Precise answers to a number of complex issues have to be addressed, such as when and where to apply a transformation, how transformations interact, what side-effects are generated and what the global effects are of a locally applied transformation. Large transformation sets also impose numerous complex requirements on the data structures used for storing the CDFG.

(2) How complete is the set of already implemented transformations?

More than a hundred transformations have been proposed in the compiler community alone. It is more than likely that the number of transformations, which can be defined for high level synthesis, is substantially higher, this due to the additional degrees of freedom, the multi-dimensionality of the cost function and the demand for high quality solutions. The nature of computation in an ASIC design is, furthermore, such that significantly more parallelism is generally available, compared to general purpose computation. This results from the fact that most ASICs (such as real time systems) operate on a semi-infinite stream of data and temporal parallelism is hence readily available.

Taking into account all the mentioned points, the important question when considering the introduction of a new transformation is one of diminishing return: do its benefits outweight the algorithmic and software implementation efforts?

(3) What are the limits on the effectiveness of transformations?

As already demonstrated in several compilers and a few high level synthesis systems and illustrated extensively later in the paper, it is not rare that a transformation can result in an order of magnitude improvement in performance. The excitement about this spectacular improvement immediately triggers the following question: what is the real limit on the power of a transformation or set of transformations for a given algorithm. While several techniques have been developed to estimate the effect of the application of other high level synthesis tasks, only simplistic, overly optimistic or pessimistic techniques have been proposed for transformations. It is our belief that no major progress on this difficult and important topic can be expected in the immediate future.

Transformations in High Level Synthesis

The current state-of-the-art in transformations can be traced over several application areas: operational research, software compilers and in particular software compilers for parallel computers, CAD, DSP ASIC and systolic array design, theoretical computer science, and numerical analysis and algebra.

A detailed overview of the transformations and their use in the mentioned fields can be found in [Pot91a]. We will restrict ourselves to an overview of the application of transformations in the high level synthesis domain.

Applying transformations in the high level synthesis process was first proposed in the late seventies [Sno78, Cas80, Mir79, McF83]. The most comprehensive sets of transformations are described in [Tri87, Wal89, Har89a, Bha90, Rab91]. Flamel [Tri87] uses five block level transformations (essentially 2 variants of loop merging and 3 forms of loop unrolling) followed by a greedy application of height reduction and constant folding. The goal is the minimization of the implementation time under area constraints, which is significantly simpler task than the problem where the hardware cost is optimized while time is the constraint. The reason for this is that in the former formulation a hierarchical problem can be solved optimally by optimizing each of the sub-functions individually. Although the optimization strategy is simple and at least partially greedy, impressive experimental improvements are achieved on 15 small examples.

The System Architect's Workbench (SAW) [Wal89] uses in-line expansion, dead code elimination, four types of selects (if-then-else transformations where the code is moved across boundaries imposed by the control structure) as well as pipelining mainly in order to support behavioral and structural partitioning. SPAID [Har89b] proposes the independent use of retiming (and pipelining as the special case of retiming), interleaving, replacement of multiplication by a constant with add/shifts and algebraic transformations. Both SAW and SPAID are interactive frameworks where designer manually explores the design trade-offs using predefined transformation mechanisms. Hi-PASS, a CAD system for DSP architecture synthesis, efficiently uses bit level retiming in designs with no hardware sharing [Dun92].

Gyrczyc [Gyr84] was one of the first to discuss the effect of loop folding. Several high level synthesis researches combine a few transformations (most often loop unrolling, software pipelining and pipelining) with scheduling. For example, Devadas [Dev89] combines scheduling with dynamic partial unwinding (unrolling) and functional pipelining with allocation, assignment, and scheduling using simulated annealing algorithm. Goossens successfully treats [Goo90] both single and nested loop folding (software pipelining) during scheduling. Hwang combines functional pipelining and loop winding with scheduling [Hwa91].

Besides the use of transformations in high level synthesis systems for computationally intensive computations and microprocessors, their successful application in the design of control dominated machines has recently been reported [Wol91a].

Pipelining is by far the most often applied transformation [Par88, Pau89]. Hartley combines pipelining and tree-high reduction (a special case of associativity), but his technique is applicable only to examples without feedbacks [Har89b]. Although pipelining is very powerful, it is not a transformation in the strict sense, since it increases the algorithm latency. Its application domain is also often limited to non-recursive algorithms [Mes88].

Mechanisms of Transformations

Although transformations can be used to optimize a wide variety of objective functions using an unlimited number of optimization approaches, transformations belong normally to a small set of classes.

HYPER currently uses three types of transformations, according to their scope:

(1) suboperational level transformations

(2) basic block transformations

(3) control structure transformations.

The first type is based on the application of knowledge about the relationship between the various operations in their hardware implementation, as well of internal structure of the computation. Very few transformations of this type are addressed in literature [Mas87, Dal89], and the most important transformation of this type in the DSP ASIC domain is substitution of multiplication by add/shifts. This transformation has been used regularly in HYPER, often with a significant improvements in both area and speed. When substituting multiplications by add/shifts, HYPER uses the Canonical Signed Digit representation to minimize the number of newly introduced operations.

The basic block transformations do not alter the control structure of a program and consist mainly of algebraic manipulations. This is in contrast with the third class, which modifies the structure and dependence of control operations such as conditionals, loops and subroutines. Since the number of combinations for both of them is unlimited, the number of transformations can be arbitrarily large. So, finding an effective and efficient subset is an important decision during the development of a high level synthesis system.

Basic Block Transformations The majority of the numerically intensive computations can be interpreted within an algebraic structure, called a *field*. A field is a relatively simple structure, consisting of two basic operations defined with the aid of a few axioms. Those operations are most often called *addition* (denoted by +) and *multiplication* (denoted by *). In addition to the fact the field is closed for both addition and multiplication (i.e. for every a, b in the set, both c = a + b and d = a * b are in the set.), the following properties are, per definition, valid:

(1) Both addition and multiplication are associative operations.

(2) Both addition and multiplication are commutative operations.

(3) The distributive law is valid (i.e. a * (b + c) = a * b + a * c).

(4) Both addition and multiplication have identity elements. These elements, usually denoted by 0 and 1, are such that for any element a in the set the following statements are valid: a + 0 = a and a * 1 = a.

(5) Both addition and multiplication have inverse elements. (For each element a in the set, there exist elements b and c in the set, such that a + b = 0, and a * c = 1.)

Although a field is simple structure, it is also very powerful. A field is a set in which one can do “additions”, subtractions”, “multiplications”, and “divisions”. The most commonly used instances of fields are: R, the set of real numbers, C, the set of complex numbers, and Q the set of rational numbers. The vast majority of the computations in DSP and other numerically intensive applications are performed in some field.

The application of algebraic transformations for the optimization of performance measure almost always translates into a computationally intractable optimization problem. To address this computational problem, a novel probabilistic sampling algorithm was introduced in HYPER. It uses the five mentioned axioms as basic moves. A crucial element of any optimization problem is the qualification of the cost function, which is both accurate and easy to compute. While this is relatively simple when the goal is to minimize the time, it is substantially more complex when the area is the measure to be minimized. An experimentally derived and statistically verified objective function has, therefore, been introduced to correlate the properties of a DCFG with the area of the final implementation. To simplify the number of moves, the associativity and inverse element law have been combined into a *generalized associativity* move.

One should be aware of some important side-effects, related to algebraic transformations such as associativity and distributivity. As a result of the finite word length representations, those transformations can result in deviating or even non-correct solutions, when applied on actual hardware. A careful analysis of the obtained result (most often using fixed point simulation) is hence necessary.

The associativity, commutativity, distributivity, identity and inverse element laws are sufficient to transform any computation in a field into any possible, equivalent form. However, as even the high school algebraic experience implies, the explicit use of some composite transformations (algebraic theorems) can greatly help to achieve better structures faster. This is the reason why the basic transformation set is often augmented with other transformations, such as factoring and common subexpression elimination and replication, in compilers and high level synthesis tools. We are currently investigating which small set of algebraic theorems is amenable as a sufficient basis for transformations.

Closely related to the above is the observation that the goal of the majority of the computations is not just to compute one output, but a set of outputs. This makes the identification of redundant and unnecessary computations non-trivial. To handle those situations, a few more transformations come in useful. The most important transformations in this class are the common subexpression elimination and replication, constant propagation, dead code elimination and strength reduction [Aho77]. The treatment of common subexpresions is a difficult and involved problem, where the decision to replicate of eliminate a common subexpression is based on a trade-off between throughput and resource utilization [Pot92b]. While the optimal treatment of constant propagation and dead code elimination is an undecidable problem in the general case, a greedy algorithm often works well in DSP practice. An optimal solution is obtained when no conditionals are present. HYPER applies the greedy algorithm after the application of any other transformation to remove the dead code and the manifest expressions, generated by those manipulations. Strength reduction is often advocated in conjunction with loop transformations and address calculation, where it is often generalized to value numbering [Wai84].

At the border between basic block and control structure transformations exists another important transformation, called *retiming*. Retiming is a conceptually simple, yet powerful transformation, which has been successfully applied in several CAD areas. It is formally defined as the distributivity property of the delay operator over most other operators: when D is defined as the delay operator, the statement D(a) * D(b) is equivalent to D(a * b) and vice versa (where * is an arbitrary operation). More informally, the retiming transformation moves delays in a CDFG such that a particular objective function is optimized. Most often either the critical path or area of the implementation is targeted. When the former case, the Leiserson-Saxe retiming algorithm generates the optimum solution in polynomial time [Lei91]. However, when area minimization is the goal, the optimization problem is NP-complete [Pot90]. Retiming is naturally defined in the form of a basic move (being a distributivity move), it operates on the same level of granularity as the algebraic transformations and acts as an enabling and disabling transformation for those algebraic transformations. HYPER therefore treats retiming using the same framework, optimization algorithm and objective function. A detailed description of the treatment of retiming in HYPER is presented in [Pot91b].

Closely associated with retiming is the *pipelining* transformation, probably the most popular high level synthesis transformation. Although pipelining is not a transformation in a strict sense as the temporal relationship between the outputs and inputs is changed, it is often used due to its capability to reduce the critical path in a graph with no feedback. Pipelining is easily related to retiming by the following observation: "Pipelining with *N* stages is equivalent to retiming where the number of delays on all inputs or all outputs, but not both, is increased by *N*".

HYPER provides separate optimization mechanisms for pipelining with a fixed and a flexible number of stages, and for various performance goals, such as throughput, area and power.

Control Structure Transformations Control structure transformations restructure the control flow of a computational graph, as implied by loop, conditional and subroutine structures. In this area, most of the attention has been devoted to the exploration of loop transformations, as it is widely recognized that this is where most of the parallelism is embodied. Therefore, this section will solely concentrate on this subject.

As in the case of the basic block transformations, we tried to identify a small set of "axiomatic" loop transformations, from which all others (or at least the majority of them) can be constructed. The two major issues in the composition and the transformation of a control structure are (i) the partitioning of the program into control sections and (ii) the determination of the temporal relationship between those units. A simple study indicates that the following transformations are sufficient to support the partitioning part: loop unfolding (and its inverse transformation, loop folding), loop fusion (and its inverse, called loop fission), loop blocking, loop permutation, loop retiming and invariant code motion.The ordering of the partitioned units is effectively supported by the software loop pipelining and loop interleaving transformations. A detailed description of all the mentioned transformations can be found in several standard compiler references [Aho77, Fis88]. It appears that the small set of transformations, mentioned above, is sufficient to support the wide spectrum of loop transformation, proposed in compiler literature.

At present, HYPER supports only partial and full loop unfolding, loop expansion and software retiming and pipelining. The development, implementation and support of other control structure transformations is under investigation.

Transformation Ordering

From previous applications in the areas of software compilation, logic synthesis and high levels synthesis, it was learned that the ordering of transformations is often the key factor in achieving their full potential. At each step of the optimization process, the system must select a transformation which is both legal and desirable. While the application of individual transformations has received a lot of attention and has yielded impressive results, barely any progress has been achieved in the area of transformation ordering. By far the most popular and widely used approach is a static ordering where the order of transformations is given a priori, most often in the form of a script. Script development is almost always based on experience and extensive experimentation. This method has at least two drawbacks: it is a time consuming process which involves a lot of experimentation and the quality of the solution is strongly dependent upon the match between the application characteristics and the underlying strategy of the script. For well defined application areas, such as logic synthesis or linear digital signal processing, this approach has proven to be very effective.

Another, often advocated, approach, is the "generate and test" technique [Wol91a]: explore all possible combinations of transformations. Although this method can be augmented with suitable implicit enumeration techniques, its obvious drawback is a large run time, exponential in the number of transformations. Another interesting approach, proposed recently, is to use a mathematical theory behind the ordering of some transformations. However, this method is limited only to a class of loop transformations [Wol91a].

HYPER supports two modes for operation ordering at present: the user guided mode and, the above mentioned, optimal scripts. In the user guided mode, the synthesis manager provides information to the user with respect to the bottlenecks in the design, their nature and the estimated effect of a transformation. A detailed description of this mode is given in [Rab90].

For several important sets of transformations and application domains, we achieved in defining simple and optimal transformation orderings, such that a particular cost function is minimized (or maximized) in polynomial time. This was, for instance, the case for the important class of linear computations over a set of seven transformations (distributivity, associativity, commutativity, inverse element law, identity element law, dead code elimination and constant propagation) [Pot92b]. Even when unfolding is added to this class, we proved that an optimal script can be defined [Pot92b]. Currently, we are addressing the design of optimal scripts for other application areas and even wider sets of transformations.

What Next in Transformations?

After been studied for several centuries in mathematics and a few decades in the compiler domain, transformations are continuing to be an important research topic in those areas. Considering the considerably shorter history of transformations in high level synthesis, it is certain that they will be a topic of intense study for some time to come.

In the area of numerically intensive applications, we expect that loop transformations supported by a loop dependency analysis (also called LCD) will be the primary tool to achieve an extra order of magnitude of performance improvement. The current probabilistic optimization algorithms, implemented in HYPER to support basic block transformations, can be sped up when supplemented with heuristics to target directly critical parts of the restructured CDFG. Increasingly background memory and I/O management and design will be supported and optimized using transformations, which directly target this type of resources.

Numerically intensive applications are being merged increasingly with non-numerical applications. A set of axiomatic and theorems based transformations, which target non-numerical transformation will therefore become important. Important elements of this class are transformations related to full and partial ordering, set membership and string manipulation. We expect that transformations which target control dominated designs will gain popularity and prove their effectiveness.

The ordering of transformations and an even more close integration and combination with other high level synthesis and software compiler tasks [Gue92] will have a significant impact on the field. Finally, and we believe most importantly, transformations will evolve and serve as the basis for of a new field of algorithm design, selection and tuning. In the next section, we will demonstrate how the combination of algorithm selection and optimizing transformations can produce results which cannot be anticipated by even advanced human designers.

ALGORITHM SELECTION AND PERFORMANCE IMPROVEMENT USING TRANSFORMATIONS

Motivation

Even a superficial, brief survey of the algorithm design research in several important application areas (such as numerical analysis, DSP, communication and information retrieval) reveals that a large variety of algorithms exist to address a single problem.

Consider, for example, the problem of sorting. In addition to the conceptually simple selection, insertion and bubble sort, there exist several more subtle algorithms, such as quick sort and Batcher bitonic sort [Knu73]. For common matrix transformations, such as fast Fourier transform (FFT) and discrete cosine transform (DCT), which are often used in audio and video processing and many other digital processing areas, there exist dozens of other fast algorithms [Nus82, Rao90]. A variety of algorithms exist to solve a Toeplitz set of equations, which occurs in a large number of signal processing applications such as for instance speech coding. The most popular ones are the Levinson-Durbin, Trench, Barlekamp-Massey and Barlekamp-Massey recursive algorithms [Bla85].

In some cases, there is an obvious superiority of one algorithm over others in terms of performance, in particular if one is interested only in asymptotic computational complexity [Gar79]. More often the optimality of an algorithm depends upon additional constraints. Although the selection of the right algorithm for a targeted task has a major impact on the final quality of the design, this area is currently more art than science or engineering and designers almost exclusively rely on intuition instead of accurate quantitative procedures. This intuitive approach is however bound to break down.

Progresses in technology have increased the application complexity and have made parallel processing a more viable alternative. Selecting the right algorithm within this setting is becoming increasingly more difficult for a human designer.

Algorithm Selection and Tuning Using Transformations

Transformations are an efficient and powerful tool for the enhancement and exploration of parallelism. It has been shown in previous section they can have a dramatic influence on the quality of an implementation. However, their effect is obviously constrained by the algorithm's computational structure. For example, it is simple to construct program instances, which are not amenable at all to transformations. Therefore, the only tools with more potential than transformations are located in the area of algorithm design and selection.

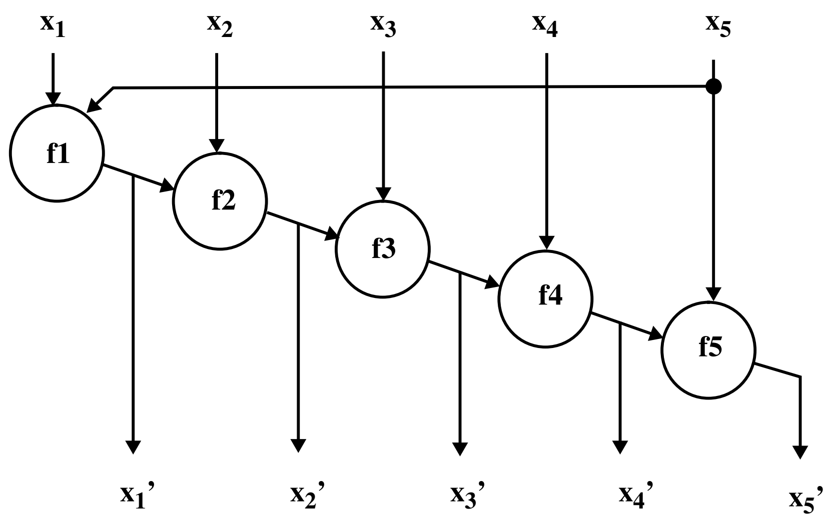
An obvious goal in this area is the automatic generation of algorithms (which adhere to a given set of specifications), such that the resulting computational graph can be easily transformed into a final implementation with maximal performance or minimal cost. It is very likely that this goal will remain elusive for years to come. However, it appears that there exist several ways to do efficient algorithm design for restricted application domains.

For instance, the algorithms used in wide classes of applications have a substantial freedom in their computational structure. This makes them prime candidates for a search to determine, which structures possess the most parallelism or hardware potential. Consider for instance the class of algorithms, which address the iterative solution of sets of equations. Examples of such algorithms are shown in Figure 2 and Figure 3. They all have the following structure:

$$x(t+1) = f(x(t)), \quad t = 0, 1, \ \ldots \tag{1}$$

where each $\boldsymbol{x}(t)$ is an n-dimensional vector and $f(.)$ is some function which has as domain and range some subset of the n-dimensional space. If the sequence $\{\boldsymbol{x}(t)\}$ generated by the above iteration converges to a *point* $\boldsymbol{x}^*$, and if the function $f(.)$ is continuous, then $\boldsymbol{x}^*$ is a fixed point of f, which satisfies the relationship $\boldsymbol{x}^* = f(\boldsymbol{x}^*)$. For example, iterative methods are often used for the solution of sparse system of equations, or for the maximization and minimization of a function by search for the zeroes of the derivatives.

When all the components of $\boldsymbol{x}$ are updated simultaneously, the method is often called the Jacoby or Gauss-Jacoby iteration. An alternative approach



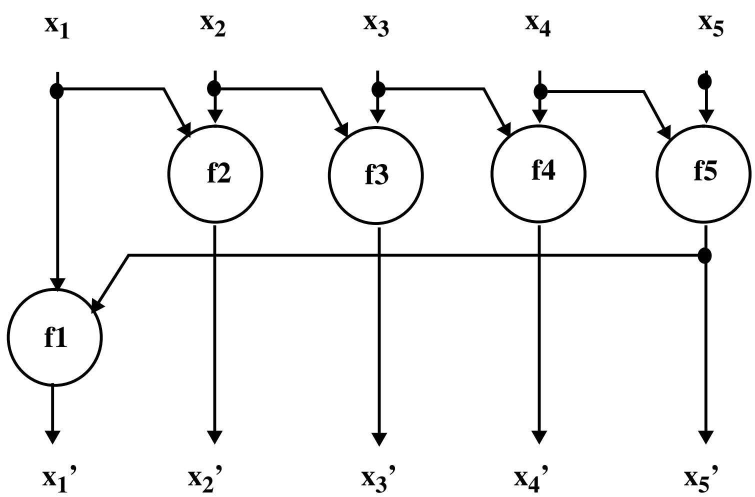
x1' = f1 (x5, x1)
x2' = f2 (x1', x2)
x3' = f3 (x2', x3)
x4' = f4 (x3', x4)
x5' = f5 (x4', x5)

FIGURE 2: Parallelism of Gauss-Seidel iterations: The initial updating order.

is to update one equation at a time. Equation (1) can then be expressed in the following form:

$$x_i(t+1) = f_i(x_1(t+1), ..., x_{i-1}(t+1), x_i(t), ..., x_n(t)), \quad i = 1, ..., n \quad (2)$$

This type of iteration is often called the Gauss-Seidel iteration and is illustrated in Figure 2 for one particular problem instance. Since the Gauss-Seidel algorithm incorporates the most recent information at each step [Ort90], it often converges faster than Gauss-Jacoby iterations. It is easily recognized that the Gauss-Seidel algorithm, due to its recursive nature, is not well suited for pipelining. Furthermore, no parallelism is available. On the other hand, it is very amenable to transformations or manipulations which address the fast execution of recursive programs to the fast implementation of recursive program transformation. This observation can be substantiated in the following way:



$$x_2' = f2\ (x_1, x_2)$$
$$x_3' = f3\ (x_2, x_3)$$
$$x_4' = f4\ (x_3, x_4)$$
$$x_5' = f5\ (x_4, x_5)$$
$$x_1' = f1\ (x_5', x_1)$$

FIGURE 3: Parallelism of Gauss-Seidel iterations: The increase in parallelism and the reduction of the critical path due to the change in the updating order.

In the Gauss-Seidel iterative algorithm, the order of updating is not fixed. Instead of starting from x_1 and proceeding forward, we can permute the updating order, as shown in Figure 3. Of course, this results in a new algorithm with different convergence properties. Nevertheless, for many sets of equations, all Gauss-Seidel algorithms will converge to the same fixed point after a number of iterations. By analyzing the speed of convergence, the performance and the amount of available parallelism, we can, for a particular application instance, construct the algorithm, which is the best suited for implementation. For example, it is easy to see that the algorithm of Figure 3 has a significantly shorter critical path than the algorithm in Figure 2 and will result in a substantially faster implementation. Besides the updating order, other aspects of the Gauss-Seidel algorithm are prone for optimization as well. For instance, one can consider the effect on both performance and convergence of the updating more than one component of $\boldsymbol{x}$ at a time.

Similar algorithm design selection techniques can be used in a variety of other numerically intensive algorithms. They include algorithms for the solution of ordinary differential equations (such as the Runge-Kutta adaptive size step methods, the Bulirsh-Stoer method, and the Predictor-Corrector technique), partial differential equation solvers and various minimization and maximization algorithms. One especially interesting domain is the class of probabilistic optimization methods.

From the above, it becomes obvious that, in order to be effective and realistic, an algorithmic designer needs a set of tools, which can help him to predict or analyze the speed of convergence (and other performance properties, such as the numerical behavior) and the implementation properties of a proposed algorithm. While simulation (or closed form analysis for some application classes) is the preferred medium for the former task, tools like HYPER with its estimations and transformations can help to predict and analyze the latter.

For many common tasks, as already mentioned, there exists a variety of sophisticated algorithms. The structure and the parameters of those algorithms were developed, most often manually by a human designer, in an attempt to optimize one particular (and often simplistic) objective function. This results in algorithms which perform well with respect to that parameter, but ignore all other dimensions of the implementation cost. A typical example of such an algorithm is the FFT, which was developed to minimize the number of multiplications in the DFT. This was a worthwhile goal at that time, but progresses in implementation technology have made this cost function less critical, while factors such as regularity and interconnect structure have gained importance.

We propose the following procedure, which helps a designer to select the most promising algorithm for a given problem instance over a set of goals and constraints.

Given a set of potentially useful algorithms $\wp$, the first step is to identify the so called *non-inferior* algorithms. An algorithm A is called inferior to an algorithm B, if all of the following criteria are fulfilled:

(1) Algorithm B uses fewer operations of all types than algorithm A;

(2) Algorithm B has a shorter critical path and shorter iteration bound than algorithm A;

(3) According to all other available lower bound and statistical estimations, algorithm B is always superior to the algorithm A. To evaluate these properties, we use the estimation techniques available in HYPER.

An algorithm A is then called non-inferior, if it is not inferior with respect to all algorithms in $\wp$.

For all non-inferior algorithms, we use branch and bound (B&B) techniques to select the best one for the given set of goals and constraints. During B&B, the implementation cost is estimated using the lower bound estimation techniques, available in HYPER. To reduce the impact of the initial algorithm specification, the algorithms are optimized and tuned over a set of transformations. This set of transformations is restricted to those transformations, which do not alter or degrade the numerical behavior of the original algorithm. Since HYPER employs very fast estimation and synthesis techniques, usually one afternoon is sufficient for the process of algorithm selection and tuning.

Proof of Concept Example

In this section, the potentials and the impact of the proposed algorithm selection technique will be demonstrated using an eight order Avenhaus Bandpass Filter as a driver example. It will be shown how the proper algorithm selection can result in dramatic improvements in either throughputs or area. Another important point, which is conveyed in this section is that transformations are an integral part of the algorithm selection process and that ignoring this point can result in inferior solutions or wrong decisions. Finally, it is demonstrated that algorithm selection is a complex process and that the results most often deviate dramatically from intuitive insights!

The importance of filters is illustrated by the quote from “Fast Algorithms for Digital Signal processing”, by R.E. Blahut [Bla85]. He says: “The most important task of digital signal processing is the task of filtering a long sequence of numbers, and the most important device is digital filter.” While there are many important families of non-linear filters, including homomorphic, morphological, nonlinear mean, order statistics and higher order Volterra filters, linear filters are the most widely used, because of their well studied and understood properties and their inherent simplicity, which in many situations guarantees satisfactory performance for a minimal cost. The widespread use of linear filters was a major factor in selecting them as the driver example in this paper.

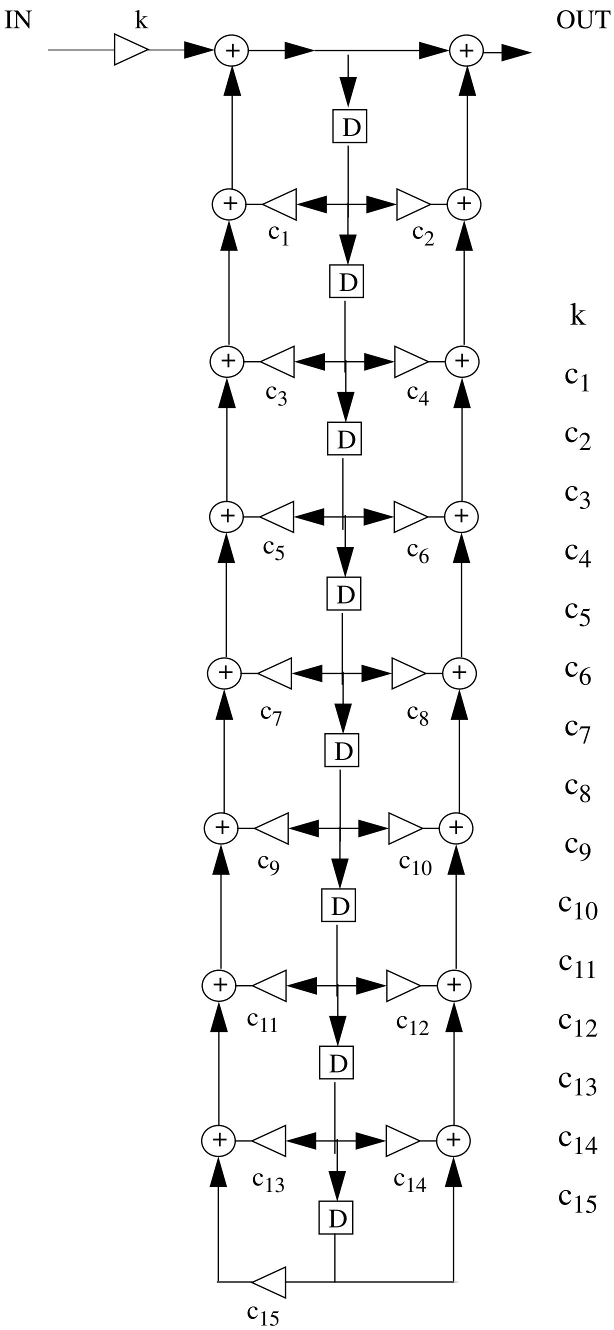
The selection of the appropriate filter structure for a specified frequency response has long been recognized as one of the crucial factors determining almost all filter characteristics such as numerical stability and implementation complexity [Cro75]. As a result, many sophisticated filter structures have been proposed. While the numerical side of the filter behavioral is well understood, the implementation complexity issue have been rarely or only marginally addressed.

The filter under study in this paper was first presented by Avenhaus [Ave72] and has often been used in the digital filter design, analysis and research. For example, Crochiere and Oppenheim [Cro75] presented in an depth discussion of several digital filter structures, which can implement the required frequency response. They compared the structures according to their statistical coefficient word length, the required number of multiplies and adds, the total number of operations and the amount of the parallelism and serialism. Although their analysis and presentation is prototype example of excellency in research, we will show that the presented measures are far from being sufficient to select a proper structure for ASIC implementation. Actually, our analysis of the example implicates that this manually conducted procedure often leads to misleading conclusions.

We consider the following five structures of the Avenhaus filter, designed by Crochiere and Oppenheim [Cro75]:

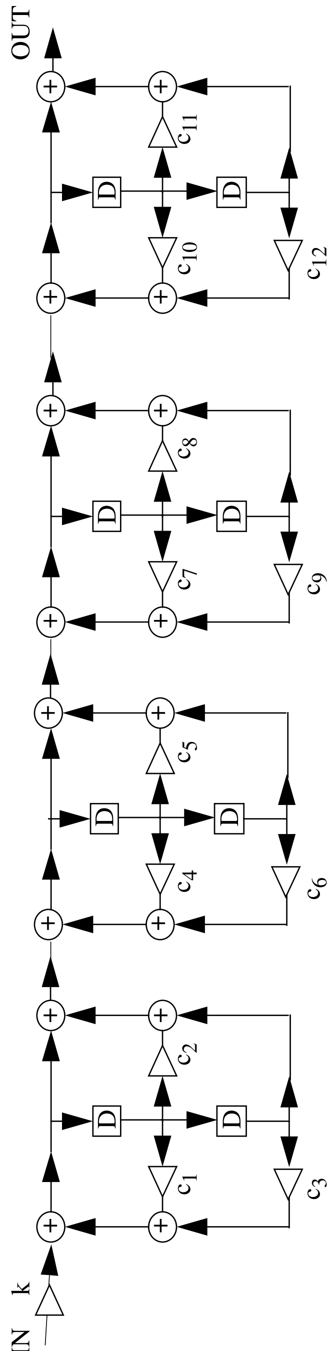
- (i) direct-form II;
- (ii) cascade form composed of direct form II sections;
- (iii) parallel form;
- (iv) continued fraction expansion structure type 1B;
- (v) Gray and Markel's ladder structure.

The corresponding structures with the appropriate coefficient values for the filter at hand are shown in Figures 4 throught 8.. The first four structures correspond to different representations of the same transfer function. The first three of them (direct-form II, cascade and parallel form) are well known and were considered very early in the filter design literature [Opp89]. The direct form corresponds to the representation of the transfer function as a ratio of polynomials, the cascade form represents the transfer function as a product of second order polynomial ratios, while the parallel and continued fraction forms correspond to partial and continued fraction expansion respectively. The continued fraction representation was first proposed by Mitra and Sherwood [Mit72, Mit73]. The fifth structure is proposed by Gray and Markel and



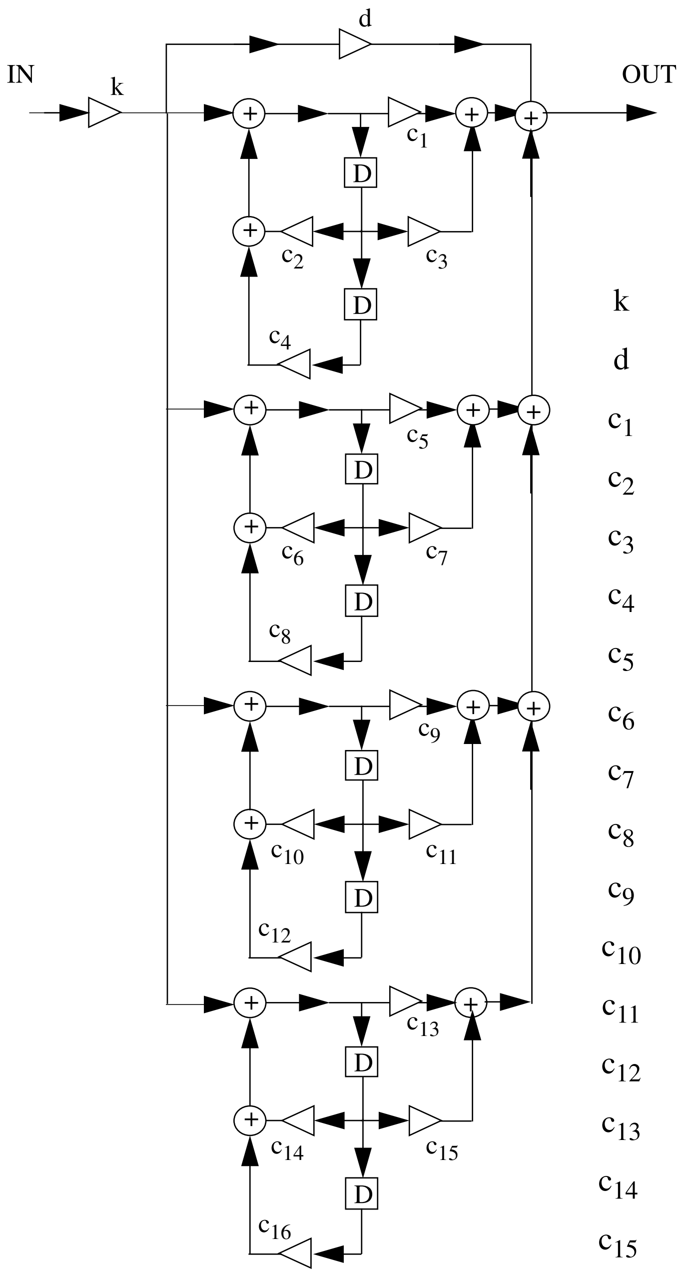
k	=	0.005656462366
c_1	=	1.479757145
c_2	=	-1.387818861
c_3	=	-4.548129731
c_4	=	3.990834175
c_5	=	4.352241822
c_6	=	-3.801090558
c_7	=	-6.802822262
c_8	=	5.978059098
c_9	=	4.094876357
c_{10}	=	-3.801090558
c_{11}	=	-4.026604318
c_{12}	=	3.990834175
c_{13}	=	-0.7833447265
c_{14}	=	-1.387818861
c_{15}	=	1.232007442

FIGURE 4: Direct-form II Structure



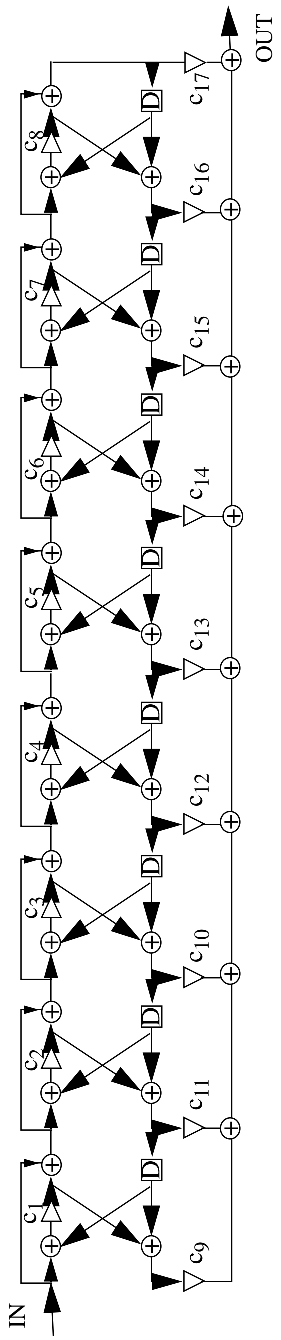
k	=	0.005656462366
c_1	=	0.2855823838
c_2	=	0.4512591755
c_3	=	-0.9116860618
c_4	=	0.4457342317
c_5	=	-1.09869455
c_6	=	-0.913078673
c_7	=	0.196901665
c_8	=	-0.0099665157
c_9	=	-0.9695353354
c_{10}	=	0.5515388641
c_{11}	=	-0.7304169706
c_{12}	=	-0.9705899009

FIGURE 5: Cascade-form (direct-form II sections) Structure



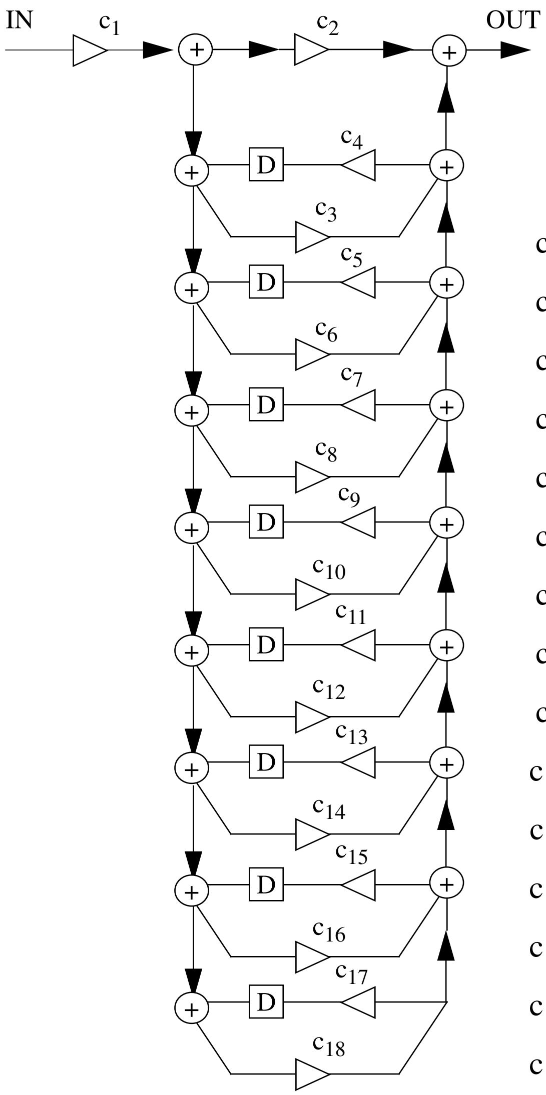
k	=	0.10000
d	=	0.07220910762
c_1	=	0.2034508401
c_2	=	0.2855823838
c_3	=	0.9026252904
c_4	=	-0.9116860616
c_5	=	0.182701196
c_6	=	0.4457342317
c_7	=	-0.9457352763
c_8	=	-0.913078673
c_9	=	-0.2052671178
c_{10}	=	0.196901665
c_{11}	=	-0.2413054652
c_{12}	=	-0.9695353354
c_{13}	=	-0.1965294023
c_{14}		0.5515388641
c_{15}		0.298888793
c_{16}		-0.9705899009

FIGURE 6: Parallel-form structure.



c1	=	0.7833447265
c2	=	-0.1885428229
c3	=	0.9843741951
c4	=	-0.1920962931
c5	=	0.9936376827
c6	=	-0.1890252997
c7	=	0.9914182038
c8	=	-0.1964790194
c9	=	0.005656462366
c10	=	0.0002916124024
c11	=	-0.001699657657
c12	=	-0.06980454614
c13	=	0.203123313
c14	=	0.01061747099
c15	=	-0.01641846145
c16	=	-1.031187912
c17	=	0.8446328267

FIGURE 7: Ladder structure by Gray and Markel.



$c_1 = 0.0002828231183$

$c_2 = 25.53154355$

$c_3 = -0.2117906668$

$c_4 = 1.017015349$

$c_5 = 0.1736820909$

$c_6 = -12.7837888$

$c_7 = 0.4649449432$

$c_8 = -1.606209516$

$c_9 = -0.2767502193$

$c_{10} = 7.610142205$

$c_{11} = 3.939247317$

$c_{12} = -0.4301416747$

$c_{13} = -0.9905202523$

$c_{14} = 0.6779657016$

$c_{15} = -21.82668197$

$c_{16} = -0.4193825017$

$c_{17} = 0.2481853405$

$c_{18} = 0.4028556869$

FIGURE 8: Continued-fraction expansion structure type 1B.

is based on an exploration of the relationship between two port networks in the analog circuit and digital filtering theory [Gra73].

In addition to presented structures, there exists a great variety of others which, of course, can also be used for the realization of the required frequency response. The most popular among them, often cited for competitive characteristics, are the wave digital filters [Fet86], the state-space digital filters [Mul76], the orthogonal [Rao84] and the multi-variable digital lattice filters [Vai85] filters (which are a generalization of both Gray-Markel's and Rao-Kailaths digital filters) and finally the systolizable IIR digital filters [Lei90].

We limited our analysis to the five mentioned structures, since our goal is not to find the "ultimate" structure for the implementation of the Avenhaus eight order bandpass filter over all structures ever proposed, but to present a procedure to classify arbitrary computational structures given a set of constraints and implementation goals. Actually, the results will indicate that no such "ultimate" exists, as the optimality of a structure strongly depends upon the goals and implementation constraints

Experimental Results

All five Avenhaus structures were described using the Silage language and passed to the HYPER environment. They were simulated in both double floating point and fixed point precision in all phases of the design process to ensure that the obtained structures were in agreement with the required frequency response.

structure	number of multiplications	number of additions	statistical word length
direct form II	16	16	21
cascade	13	16	12
parallel	18	16	11
continuous fraction	18	16	23
ladder	17	32	14

Table 1: Number of operations and Statistical Word Length.

Table 1 (assembled using data from [Cro75]) shows the number of multiplications and addition and the statistical word length for all five forms. The statistical word length is the minimum number of bits needed, such that the resulting structure still meets the frequency domain specifications (as determined by a statistical analysis). We simulated all five examples, and indeed, all of them produced the required frequency response. A small correction was needed for the direct form II, as, due to a typographical error, two coefficients were interchanged in the Crochiere and Oppenheim paper.

Table 2 shows the number of operations in the five structures after the multiplication strength reduction, which substitutes all multiplications by a proper combinations of shifts and adds. Generally speaking, of course, there is a correlation between the number of multiplications and the word length in the initial form and the number of shift-operations after the expansion. However, only a precise analysis can determine the final effect of the application of the multiplication strength reduction.

structure	total	additions	shifts	subtractions	transfers
direct form II	215	58	103	46	7
cascade	94	31	40	18	4
parallel	113	33	51	24	4
continuous fraction	205	55	106	43	-
ladder	116	35	49	31	-

Table 2: Number of operations after Multiplication Expansion. Note that all structures have one input operation in addition to the listed Operations. The transfer operations are register-register transfers, needed for the implementation of delays.

For example, although the direct form has both fewer multiplications (16 vs. 18) and a shorter word length (21 vs. 23) than the continuous fraction, the latter eventually requires fewer shifts (103 vs. 92) and a smaller number of operations (215 vs. 178). From this table, it already becomes clear that, even when implementing the filter on a general purpose, single processor computer, the selection of the right algorithm is extremely important (for instance because of the wide span over the number of operations: 94 versus 215). This is even more true when considering ASIC implementations.

Table 3 shows the length of the critical path for all five forms in the initial format and after the application of retiming and pipelining. All examples are implemented using the word lengths as indicated in Table 1. We decided to use only those two transformations, because they never alter the bit width requirements. Although HYPER has more than 20 other transformations, many of them may alter the numerical behavior and hence the required data representation. Distributivity is an example of such a transformation. During the optimization for the critical path, we used the Leiserson-Saxe retiming algorithm for both retiming and pipelining [Lei83,Pot92a]. During the optimization for area, retiming and pipelining for resource utilization was used [Pot91b, Pot92b]. As will be demonstrated below, those few transformations had already profound effects on the final results.and

structure	**initial**	**retimed**	**pipelined**
direct form II	980	686	686
cascade	527	341	279
parallel	609	522	261
continuous fraction	2332	1908	1908
ladder	2835	2835	630

Table 3: Critical path for the Avenhaus Eight Order Bandpass Filter (in μsec - for a 2 μm technology).

In the initial structures, the ratio in critical paths between the fastest (cascade) and the slowest (ladder) structures equals 5.4. When no extra latency is allowed, this ratio becomes even higher (8.3) as the cascade structure is very amenable to the retiming transformation, while this is clearly not the case for the ladder filter. If we allow the introduction of one pipeline stage, the parallel form becomes the fastest structure. It is important to notice that, when throughput is the major concern, other filter structures can be conceived, which result in even smaller critical paths. For example, the recently proposed maximally fast implementation of linear computations [Pot92b] reduces the length of the critical path to only 174 nanoseconds without introducing any latency. This is the improvement by the factor of 16.3 over the original ladder structure. When additional latency is allowed, this factor can be improved to arbitrarily high levels [Pot92b].

Table 4 shows the area of the final implementation for the five structures for six different sampling periods. For all five forms, results are shown for both the original and the transformed structures. Note that the feasibility of acheiving the required performance is a function of the selected structure and the applied transformations. It can be observed that the ratio between the largest and smallest implementations is even higher than the mentioned performance ratios.

For example, for a sampling period of 1 micro-second, only the direct, cascade and parallel forms are feasible alternatives. The ratio of the implementation areas between the direct form and the cascade form equals 12.9. This ratio is improved to 15.1 for the retimed cascade form (retiming was done using the retiming for resource utilization [Pot91b]). Interestingly enough, there is not a single instance were pipelining improved the area requirements. This is the consequence of the fact that in all designs, except for the fastest implementations, the major part of the area cost is located in the registers and not the execution units or the interconnect. Most often, pipelining increases registers requirements even more. Also, due to recursive nature of the filters, the effectiveness of pipelining is limited.

structure	**sampling period (in μsec)**					
	5	**4**	**3**	**2**	**1**	**0.5**
df II I	20.32	20.32	22.88	26.36	92.49	-
df II PR	19.86	19.86	21.75	30.65	53.98	-
cascade I	6.63	6.63	6.63	6.63	8.97	-
cascade P	5.98	5.98	5.98	5.98	6.11	11.36
parallel I	6.71	6.71	6.71	6.71	8.67	-
parallel R	5.92	5.92	5.92	5.92	7.16	-
parallel P	5.92	5.92	5.92	5.92	7.16	13.42
con frac I	14.65	22.67	-	-	-	-
con fra RP	13.56	19.54	27.19	-	-	-
ladder RI	5.73	7.38	-	-	-	-
ladder P	5.73	7.38	10.77	16.87	-	-

Table 4: Implementation Area for six different throughput requirements.
I - initial structure; R - retimed structure; P - pipelined structure.

Table 5 shows the critical path for all five forms (original and transformed) when the statistical word length information is not used and all structures are implemented with 23 bit data and coefficients. All forms simulate well for this word length. It can be observed that the critical path increased for all structures, except for the continuous expansion form, which needs 23 bits in any case. For example, the critical path of the 23 bit ladder filter (5300 μsec) is 20 times higher than what was obtained for the optimal cascade or parallel structures with an optimized word length (261 μsec). The obvious and important conclusion is that the importance of word length optimization can not be overstated and is a crucial component during the optimization for both area and throughput. Preliminary studies also indicate that the importance of word length optimization is even higher when power and testability are considered. Interestingly enough, until recently, very little attention was paid to this important issue, which often has a dramatic influence on the quality of the final implementation.

Looking again at Table 4, we see that, depending upon the required throughput, the minimal area is obtained by different structures. For the fastest designs (designs where throughput rate is higher than 1 MHz), the pipelined cascade (when additional latency is allowed) and the parallel forms are the smallest solutions. For medium speeds (sampling period between 1 and 4 μs) the parallel form achieves the smallest area. Finally and most surprisingly, when the throughput requirements are the least strict, the ladder form is the most economical implementation. Notice that ladder form does neither have the smallest bit width nor the fewest number of operations. However, its regular and balanced structure requires few registers and few interconnects, which results in a slightly smaller area than other implementations.

structure	**initial**	**retimed**	**pipelined**
direct form II	1060	742	742
cascade	1113	795	689
parallel	1484	1431	689
continuous fraction	2332	1908	1908
ladder	5300	5300	1272

Table 5: Critical Path for 23 bit Word Lengths (for all structures).

It is very important to observe that the previous analysis does not indicate that any form is a-priory superior in terms of speed or area. The results depend strongly upon the required frequency response, the applied transformations, the objective function (for instance power or testability) or the performance parameters (for instance signal to noise ratio or overflow behavior). For instance, the conclusions might be different for a 7th order elliptical low-pass filter. Our point is that for a given set of goals and constraints, only a quantitative analysis such as proposed in this paper, can provide a more qualified view, which can help designers in making the proper decisions during the design process.

Finally, it is worth mentioning that all results presented in this section, if not otherwise stated, were obtained using the HYPER system. All results were generated and analyzed in a time span of two hours, which demonstrates that efficiency of HYPER is high enough to address the algorithm selection and tuning problem.

FUTURE DIRECTIONS IN HYPER

It has been a proven wisdom that the quality of an answer is almost always proportional to the number of questions it rises. At least according to this criteria, the HYPER approach has been highly successful. Design experience with HYPER have lead to a number of exciting novel design approaches and research venues. For instance, it was realized that high level synthesis techniques and optimizations can have a major impact on the power consumption of an implementation. This result is especially interesting given the current interest in portable consumer, computation and communication devices. A low power design methodology has been developed and implemented in HYPER [Cha92]. We have furthermore realized that the synthesis techniques, used in HYPER (especially the estimations, transformations and instruction and hardware selection) are amenable to other architectural styles as well. This has lead to the development of a a number of related synthesis environments for field programmable data path architectures[Che92] and multi-processor programmable DSP's [Hoang92]. We are currently investigating how HYPER can be adapted to address the retargetable compilation problem for programmable DSP's. Finally, using similar concepts as described in this paper, it can be easily seen that HYPER can be extended to address the architectural design space exploration and performance analysis problems. Efforts in this direction are under way.

Both transformations and estimations (which were not discussed here due to lack of space) are research areas in the early stages of development. Their impact in domains such high level and system level synthesis will increase with the development of novel and more powerful techniques. These results might be reflected into more traditional domains, such as retargetable compilers and compilers for massively parallel machines. It is our belief that tools such as HYPER can act as a seed for widely used, industrial strength synthesis environments. Finally, we are convinced that computer aided algorithm selection and tuning is one of techniques, which will help the designer to manage and conceive ever more complex systems and applications.

REFERENCES:

[Aho77] A.V. Aho, J.D. Ullman: "Principles of Compiler Design", Addison-Wesley Publishing Co., Reading, MA, 1977.

[Ave72] E. Avenhaus: "On the design of digital filters with coefficients of limited word length", IEEE Trans. on Audio and Electroacoustics, Vol. 20, pp. 206-212, 1972.

[Bar73] M. Barbacci: "Automatic Exploration of the Design Space for Register Transfer (RT) Systems", *Ph.D. Thesis*, Dept. of CS, Carnegie-Mellon University, November 1973.

[Bha90] J. Bhaskar, H. Lee, "An Optimizer for Hardware Synthesis", IEEE Design and Test of Computers, Oct. 1990.

[Bla85] R. E. Blahut, "Fast Algorithms for Digital Signal Processing", Addison-Wesley Publishing Company, 1985.

[Bro92] R.W. Brodersen, ed.: "Anatomy of a Silicon Compiler", Kluwer Academic Publishers, Boston, MA, 1992.

[Cam91] R. Camposano, R.A. Walker: "A Survey of high-level synthesis systems", Boston: Kluwer Academic, Norwell, Mass., 1991.

[Cas80] A.E. Casavant, D.D. Gajski, D.J. Kuck: "Automatic design with dependence graphs", 17th ACM/IEEE Design Automation Conference, pp. 506-515, 1980.

[Cha92] A.P. Chandrakasan, at. al.: "Hyper-LP: A Design System for Power Minimization using Architectural Transformations", IEEE International Conference on Computer-Aided Design, Santa Clara, CA, paper 6c.3, November 1992.

[Che92] D.C. Chen, at. al.: "An Integrated System for Rapid Prototyping of High Performance Algorithm Specific Data Paths", IEEE ASAP-92, pp. 134-148, 1992.

[Chu92] C-M. Chu, J. Rabaey: "Harware Selection and Clustering in the HYPER Synthesis System", IEEE EDAC-92, Brusses, Belgium, pp. 176-180, 1992.

[Cro75] R.E. Crochiere, A. V. Oppenheim: "Analysis of Linear Networks", Proceeding of the IEEE, Vol. 63, No. 4, pp. 581-595, 1975.

[Dal89] W.J. Dally: "Micro-Optimization of Floating point Operations" Third Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS III), SIGARCH Computer Architecture News, Vol. 17, No. 2, pp. 283-289, 1989.

[DeM92] H. De Man: "Design Technology Research for the Nineties: More of the Same?", Proceedings of the European Design Automation Conference, Hamburg, Germany, pp. 592-596, 1992.

[Dev89] S. Devadas, A.R. Newton: "Algorithms for Hardware Allocation in Data Path Synthesis", *IEEE Transaction on CAD*, Vol 8, No 7, pp. 768-781, 1989.

[Dun92] P. Duncan, S. Swamy, S. Sprouse, D. Potasz, R. Jain, N. Gafter, W. Cammack, Y. Wong, W. Gass: "HI-PASS: A Computer Aided Synthesis System for Fully Parallel Digital Signal Processing ASIC's" to appear in Proceedings of the International Conf. on Acoustic, Speech & Signal Processing, March 1992, San Francisco.

[Fet86] A. Fettwies: "Wave Digital Filters: Theory and Computation Methods", Proceedings of the IEEE, Vol. 74, No. 2, pp. 270-327, 1986.

[Fis88] C.N. Fischer, R.J. LeBlanc, Jr.: "Crafting a Compiler", The Benjamin/Cummings Publishing Co. Inc., Menlo Park, CA, 1988.

[Gar79] M.R. Garey, D.S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H.Freeman and company, New York, 1979.

[Goo90] G. Goossens, J. Rabaey, J. Vandewalle, H. De Man: "An Efficient Microcode Compiler for Application Specific DSP Processors", IEEE Transaction on CAD of integrated circuits and systems, Vol. 9, No. 9, pp. 925-937, 1990.

[Gra73] A.H. Gray, J.D. Markel: "Digital lattice and ladder filter synthesis", IEEE Trans. on Audio Electroacoustic, Vol. 21, pp. 491-500, 1973.

[Gue92] L. Guera: Personal Communications, 1992.

[Gyr84] E. Gyrczyc: "Automatic Generation of Microsequenced Data Paths to Realize ADA Circuit Description", *Ph.D. Thesis*, Carleton University, 1984.

[Har89a] R. Hartley, A. Casavant: "Tree-height Minimization in Pipelined Architectures", IEEE IC CAD, pp.112-115, 1989.

[Har89b] B.S. Haroun, M.I. Elmasry: "Architectural Synthesis for DSP Silicon Compilers", IEEE Transaction on CAD for IC, Vol. 8, No. 4, pp. 431-447, 1989.

[Hen89] J.L. Henessy, D.A. Patterson: "Computer architecture: a quantitative approach", San Mateo, Calif.: Morgan Kaufman Publishers, 1989.

[Hil92] P. Hilfinger, J. Rabaey: "DSP Specification Using the Silage Language", in "Anatomy of a Silicon Compiler", ed. by R.W. Brodersen, pp. 199-220, Kluwer Academic Publishers, Boston, MA, 1992.

[Hoa92] P. Hoang, J. Rabaey: "Hierarchical Scheduling of DSP Programs onto Multiprocessors for Maximum Throughput", IEEE ASAP-92, pp. 21-36

[Hwa91] C.-T. Hwang, Y.-C. Hsu, Y.-L. Lin: "Scheduling for Functional Pipelining and Loop Winding", ACM/IEEE Design Automation Conference, San Francisco, pp. 764-769, 1991

[Lei90] S.-M. Lei, K. Yao: "A Class of Systolizable IIR Digital Filters and Its Design for Proper scaling and Minimum Output Roundoff Noise", IEEE Trans. Circuits and Systems, Vol. 37, No. 10, pp. 1217-1230, 1990.]

[Lei83] C.E. Leiserson, F.M. Rose, J.B. Saxe, "Optimizing synchronous circuits by retiming", Proceedings of Third Conference on VLSI, pp. 23-36, Computer Science Press, 1983.

[Lei91] C.E. Leiserson, J.B. Saxe: “Retiming Synchronous Circuitry”, Algorithmica, Vol. 6, pp. 5-35, 1991

[Knu73] D.E. Knuth: “The art of computer programming, Vol. 3: Sorting and searching”, Addison-Wesley, Reading, M

[Mas87] H. Massalin: “Superoptimizer: A look at the Smallest Program”, Second Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS II), SIGPLAN Notices, Vol. 22, No. 10, pp. 122-127, 1987.

[McF83] M.C. McFarland, A.C. Parker: “An Abstract Model of Behavior for Hardware Descriptions”, IEEE Transaction on Computers, Vol. 32, No. 7, pp. 621-636, 1983.

[McF90] M.C. McFarland, A.C. Parker, R. Camposano: “The High-Level Synthesis of Digital Systems””, Proceedings of the IEEE, Vol. 78, No. 2, pp. 301-317., February 1990.

[Mes88] D. Messerschmitt, “Breaking The Recursive Bottleneck”, in *Performance Limits in Communication Theory and Practice,* Kluwer Academic Publishers, 1988.

[Mir79] G.S. Miranker: “The use of conflicts in the translation and optimization of hardware description languages”, Ph. D. dissertation, MIT, May 1979.

[Mit72] S.K. Mitra, R.J. Sherwood: “Canonic realization of digital filters using the continued fraction expansion”, IEEE Trans. on Audio Electroacoustics”, Vol. 21, pp. 185-194, 1972.

[Mit73] S.K. Mitra, R.J. Sherwood: “Digital ladder networks”, IEEE Trans. on Audio Electroacoustics”, Vol. 21, pp. 185-194, 1972.

[Mul76] C.T. Mullis, R.A. Roberts: “Synthesis of minimum roundoff noise fixed-point digital filters”, IEEE Trans. Circuits and Systems, Vol. 23, No. 9, pp. 551-561, 1976.

[Nus82] H.J. Nussbaumer: “Fast Fourier transform and convolution algorithms”, 2nd edition, Springer-Verlag, Berlin, Germany, 1982.

[Opp89] A.V. Oppenheim, R.W. Shafer: “Discrete-time Signal Processing”, Prentice Hall, Englewood Cliffs, NJ, 1989.

[Ort90] J.M. Ortega: “Numerical analysis: a second course”, Philadelphia: Society for Industrial and Applied Mathematics, 1990.

[Par88] N. Park, A.C. Parker: “Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications”, *IEEE Trans. on CAD*, Vol 7, No. 3, pp. 356-370, 1988.

[Pat89] D.A. Patterson, J.L. Henessy: “Computer architecture: a quantitative approach”, San Mateo, Calif. : Morgan Kaufman Publishers, 1989.

[Pau89] P.G. Paulin, J.P. Knight: “Force -Directed Scheduling for the Behavioral Synthesis of ASIC”, *IEEE Transaction on CAD*, Vol 8., No 6, pp. 661-679, 1989.

[Pau92] P.G. Paulin: “DSP Design Tool Requirements for the Nineties: An Industrial Perspective”, Submitted to the High Level Synthesis Workshop.

[Pit90] I. Pitas, A.N. Venetsanopoulos: “Nonlinear Digital Filters”, Kluwer Academic Publishers, Boston, MA, 1990.

[Pot91a] M. Potkonjak: “Algorithms for High Level Synthesis Resource Utilization Based Approach”, *Ph.D. Thesis*, University of California at Berkeley, 1991.

[Pot91b] M. Potkonjak, J. Rabaey: “Optimizing the Resource Utilization Using Transformations”, *Proc. IEEE ICCAD Conference*, Santa Clara, November 1991.

[Pot92a] M. Potkonjak, J. Rabaey: “Pipelining: Just Another Transformation”, IEEE ASAP-92, pp. 163-175, 1992.

[Pot92b] M. Potkonjak, J. Rabaey: “Maximally Fast and Arbitrarily Fast Implementation of Linear Computations”, IEEE International Conference on Computer-Aided Design, Santa Clara, CA, paper 6c.4, November 1992.

[Rab90] J. Rabaey, and M. Potkonjak, “Resource Driven Synthesis in the HYPER system,” *ISCAS-90*, vol. 4, pp. 2592-2595, New Orleans, LA, May 1990.

[Rab91] J. Rabaey, C. Chu, P. Hoang, M. Potkonjak: “Fast Prototyping of Data Path Intensive Architecture”, *IEEE Design and Test*, Vol. 8, No. 2, pp. 40-51, 1991.

[Rao84] S.K. Rao, T. Kailath: “Orthogonal digital filters for VLSI implementation”, IEEE Trans. Circuits and Systems, Vol. 31, No. 11, pp. 933-945, 1984.

[Rao90] K.R. Rao, P. Yip: “Discrete cosine transform: algorithms, advantages, applications”, Academic Press, Boston, MA, 1990.

[Sno78] E.A. Snow, D.P. Siewiorek, D.E. Thomas: “A technology-relative computer-aided design system: abstract definition, transformations and design tradeoffs”, 15th ACM/IEEE Design Automation Conference, pp. 220-226, 1978.

[Tri87] H. Trickey: “Flamel: A high-Level Hardware Compiler”, IEEE Transaction on CAD, Vol. 6, No. 2, pp. 259-269, 1987.

[Vai85] P.P. Vaidyanathan, S.K. Mitra: “A general family of multivariable digital lattice filters”, IEEE Trans. Circuits and Systems, Vol. 32, No. 12, 1985.

[Wai84] W.M. Waite, G.Goos: Compiler construction, New York: Springer-Verlag, 1984.

[Wal89] R.A. Walker, D.E. Thomas: “Behavioral Transformation for Algorithmic Level IC Design” *IEEE Trans. on CAD*, Vol 8. No.10, pp. 1115-1127, 1989.

[Wal91] R.A. Walker, R. Camposano: “A Survey of High-Level Synthesis Systems”, Kluwer Academic Publishers, Boston, Ma, 1991.

[Wol91a] W. Wolf, A. Takach, T-C. Lee: “Architectural Optimization Methods for Control Dominated Machines.” in W. Wolfe R. Camposano: “High-Level VLSI Synthesis”, Kluwer Academic Publishers, pp. 231-254., 1991.

[Wol91b] M. Wolfe: “The Tiny Loop Restructuring Research Tool”, 1991 International Conf. on Parallel Processing, Vol. II, pp. 46-53, 1991.